



IBM Research

The IBM High Performance Computing Toolkit

I-Hsin Chung
Advanced Computing Technology
IBM T.J. Watson Research Center
ihchung@us.ibm.com

What is it?

- **IBM long-term goal:**
 - An automatic performance tuning framework
 - Assist users to identify performance problems
 - Provide possible solutions
 - A common application performance analysis environment across all HPC platforms
 - Look at all aspects of performance (communication, memory, processor, I/O, etc) from within a single interface
- **Where we are: one consolidated package**
 - One consolidate package (AIX, Linux/Power)
 - Tools for MPI, OMP, processor, memory etc
 - Operate on the binary and yet provide reports in terms of source-level symbols
 - Dynamically activate/deactivate data collection and change what information to collect
 - One common visualization GUI

IBM HPC Toolkit Software Components

- **Hardware (CPU) Performance**
 - Xprofiler
 - HPM Toolkit
 - hpmcount
 - libhpm
 - hpmstat
- **Shared Memory Performance**
 - Dpomp, PompProf
- **Message-Passing Performance**
 - MPI profiler/tracer
- **Memory Performance**
 - Sigma
- **Performance Visualization**
 - PeekPerf
- **I/O Performance**
 - MIO (modular I/O)

Supported Platforms

- **AIX**
- **Linux**
 - PowerPC
 - Blue Gene /L and Blue Gene /P
 - Intel x86 (coming soon)
 - AMD (coming soon)
- **Windows (Intel/AMD) + Mac (coming soon)**
 - Offline Peekperf visualization capability only

AGENDA

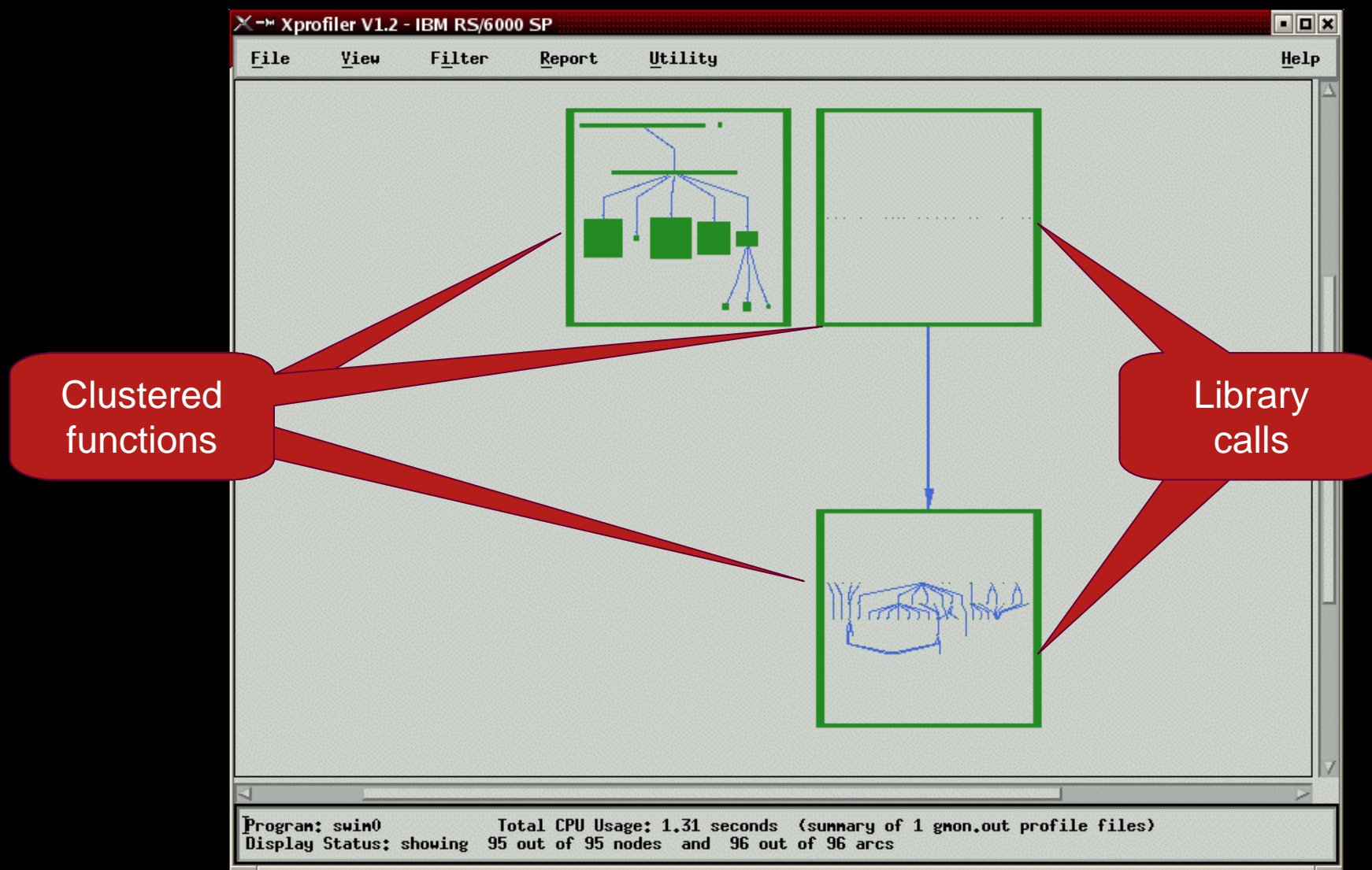
- **Xprofiler**: call-graph profiling
- **HPM**: hardware counter data
- **MPI Profiler/Tracer**: MPI profiling
- **PompProf**: OpenMP profiling
- **SIGMA**: memory profiling
- **MIO**: I/O profiling and optimization
- **IBM HPC Toolkit**
- **Questions/Comments**

XProfiler

Xprofiler

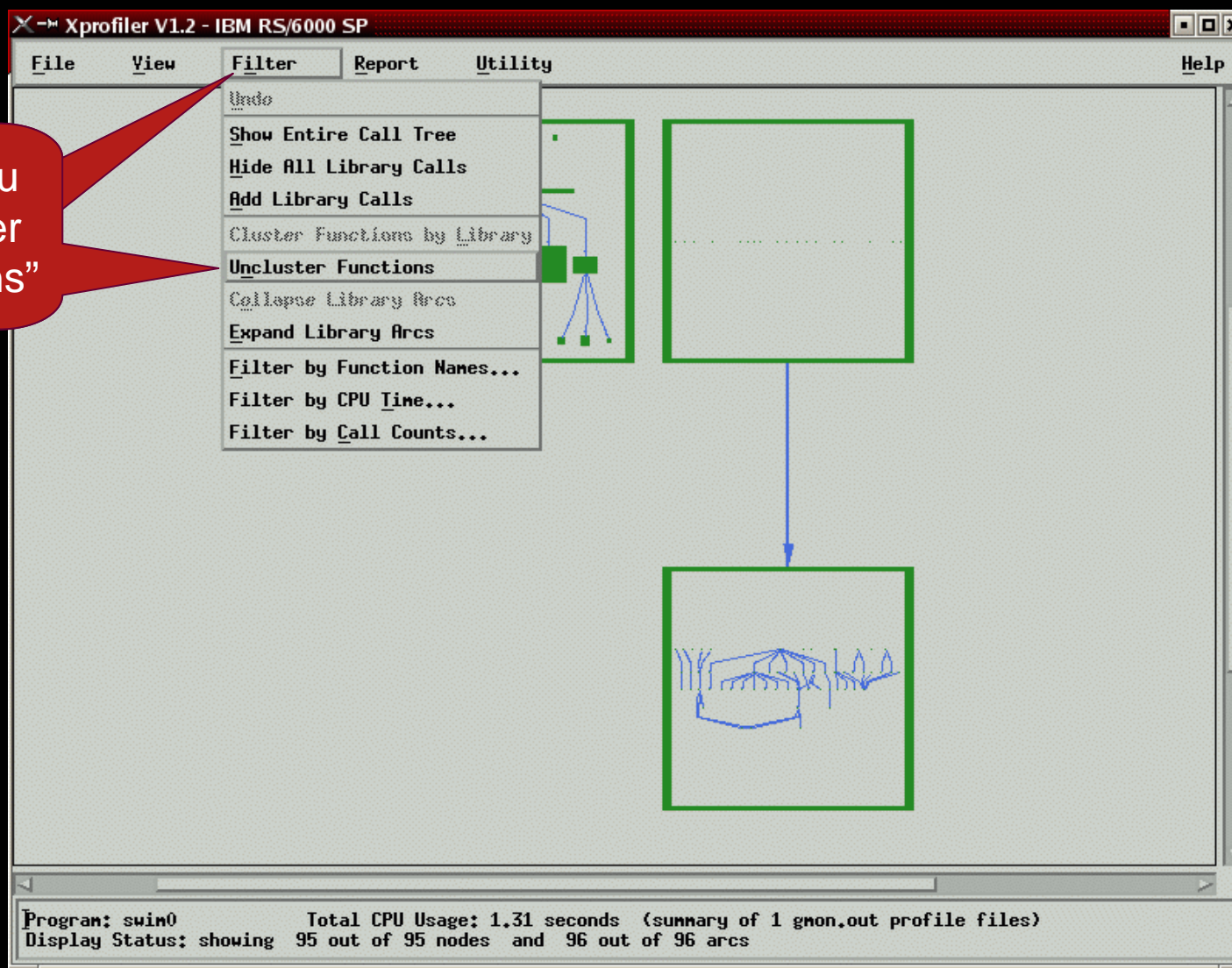
- **Visualizer CPU time profiling data**
- **Compile and link with -g -pg flags + optimization**
- **Code execution generates gmon.out file**
 - MPI applications generate gmon.out.1, ..., gmon.out.n
- **Analyze gmon.out file with Xprofiler**
 - xprofiler a.out gmon.out
- **Important factors:**
 - On AIX time-sampling interval is 0.01 sec
 - Profiling introduces overhead due to function calls

Xprofiler - Initial View

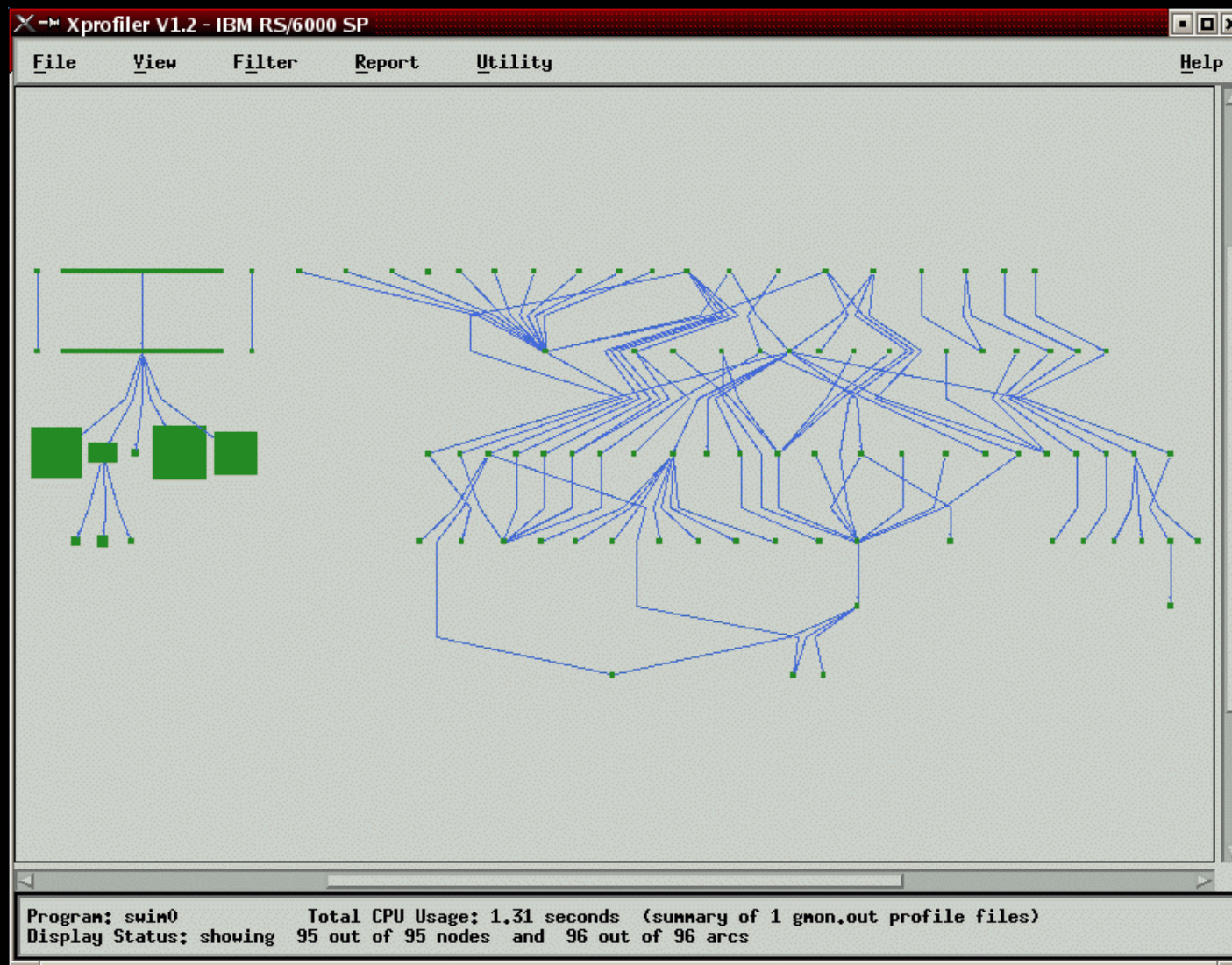


Xprofiler - Unclustering Functions

on "Filter" menu
select "Uncluster
Functions"



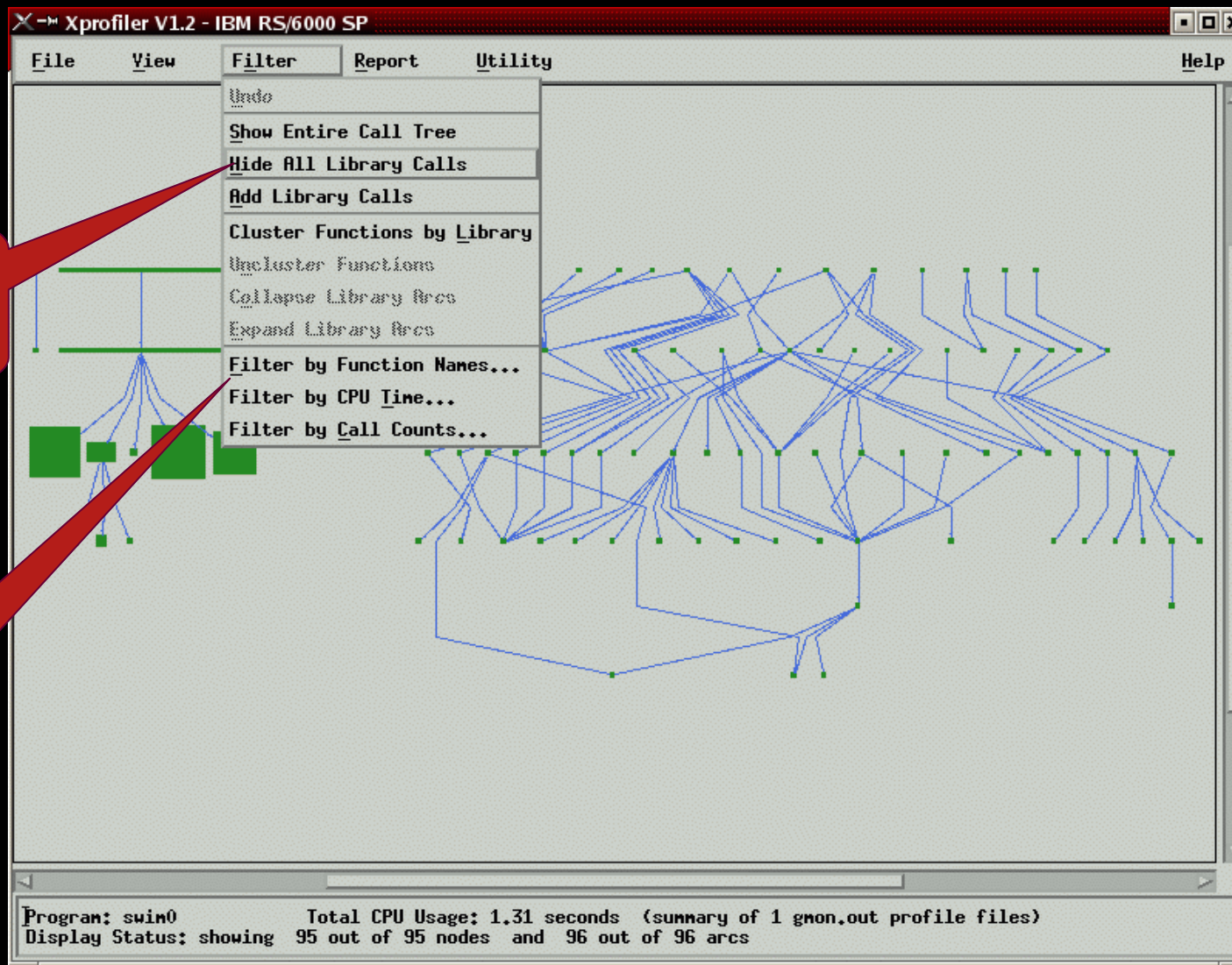
Xprofiler - Full View - Application and Library Calls



Xprofiler - Hide Lib Calls Menu

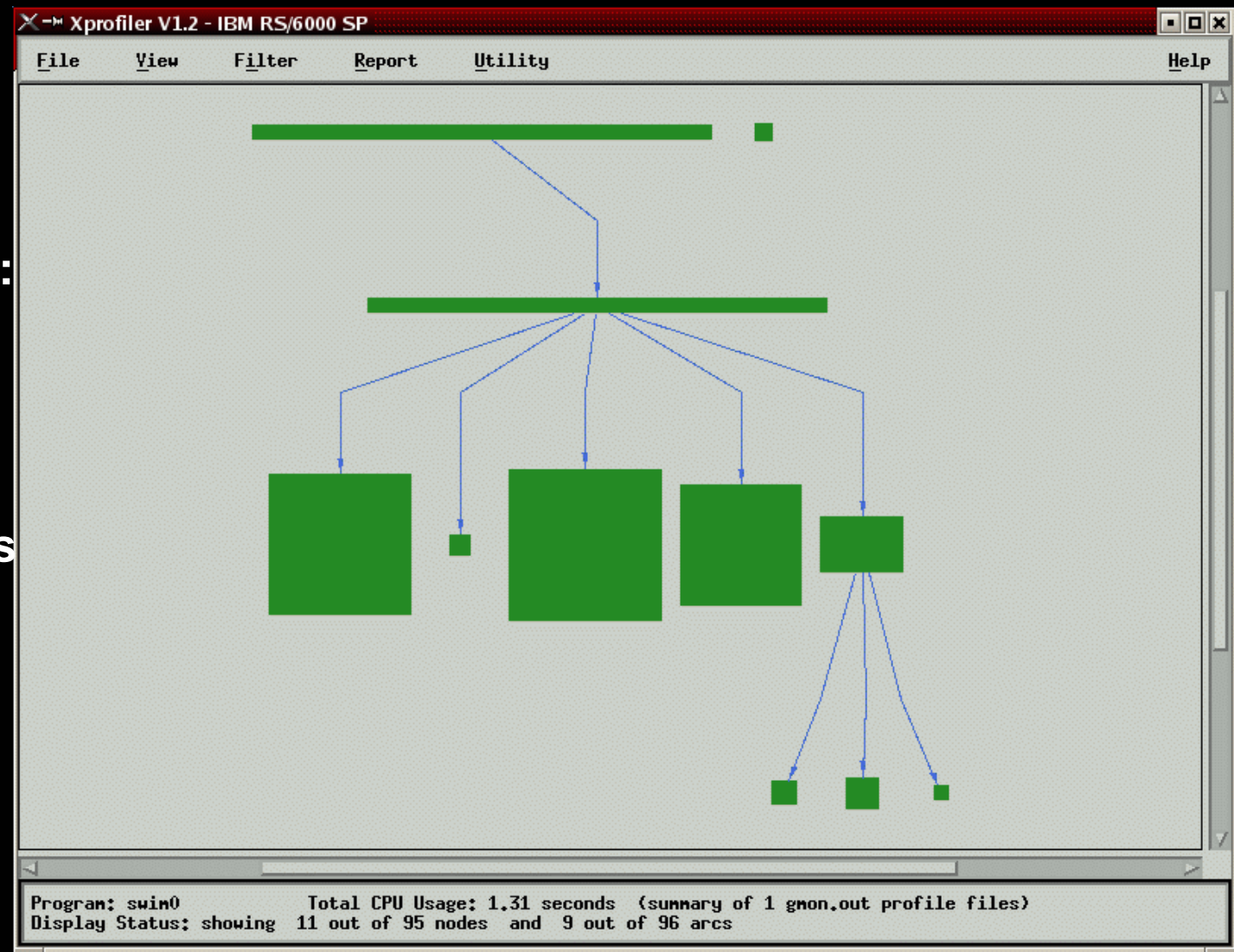
Now select
“Hide All
Library Calls”

Can also filter by:
Function Names,
CPU Time,
Call Counts

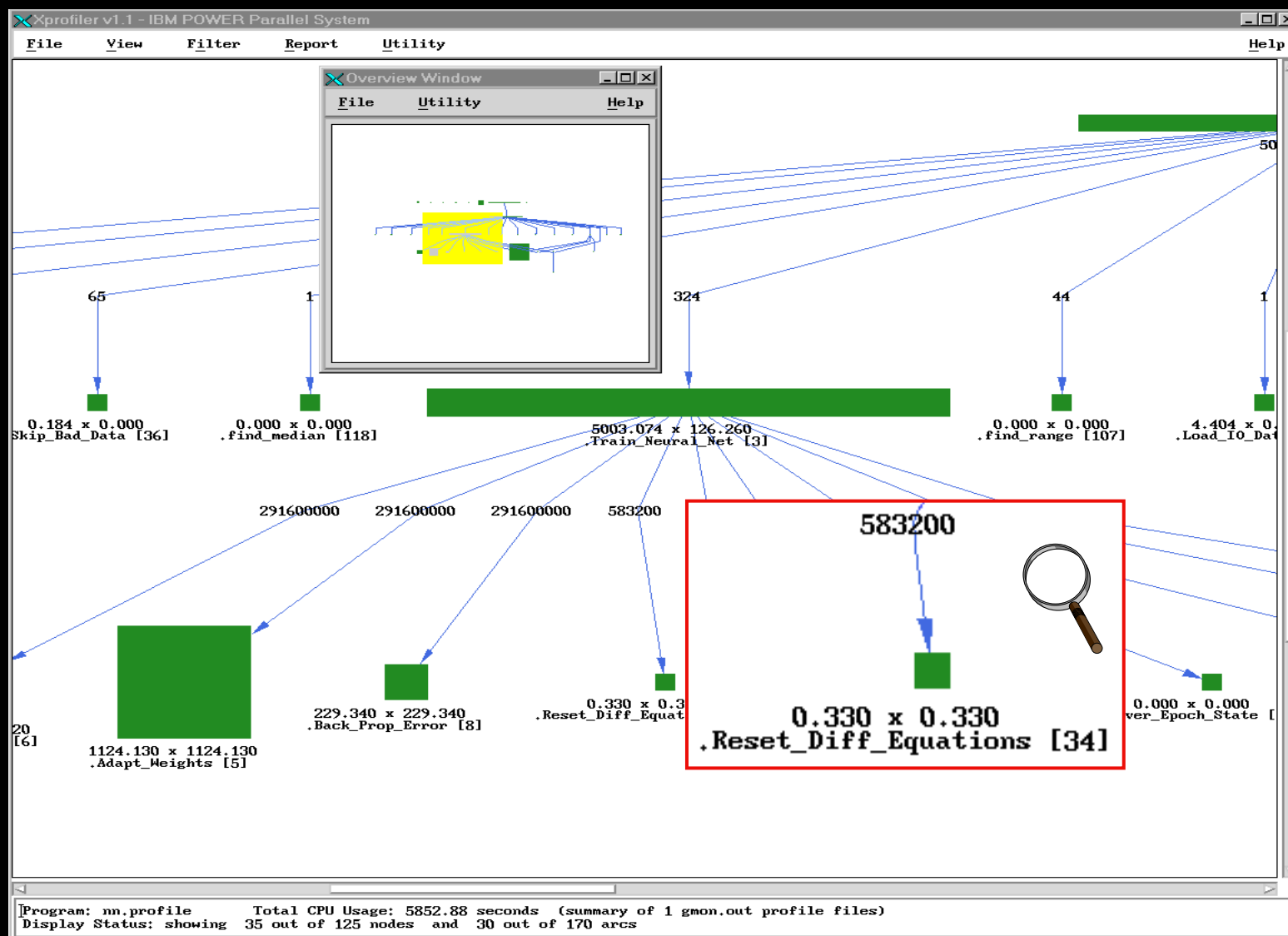


Xprofiler - Application View

- **Width of a bar:** time including called routines
- **Height of a bar:** time excluding called routines
- **Call arrows** labeled with number of calls
- **Overview window** for easy navigation (View → Overview)



Xprofiler: Zoom In



Xprofiler: Flat Profile

- **Menu Report** provides usual gprof reports plus some extra ones

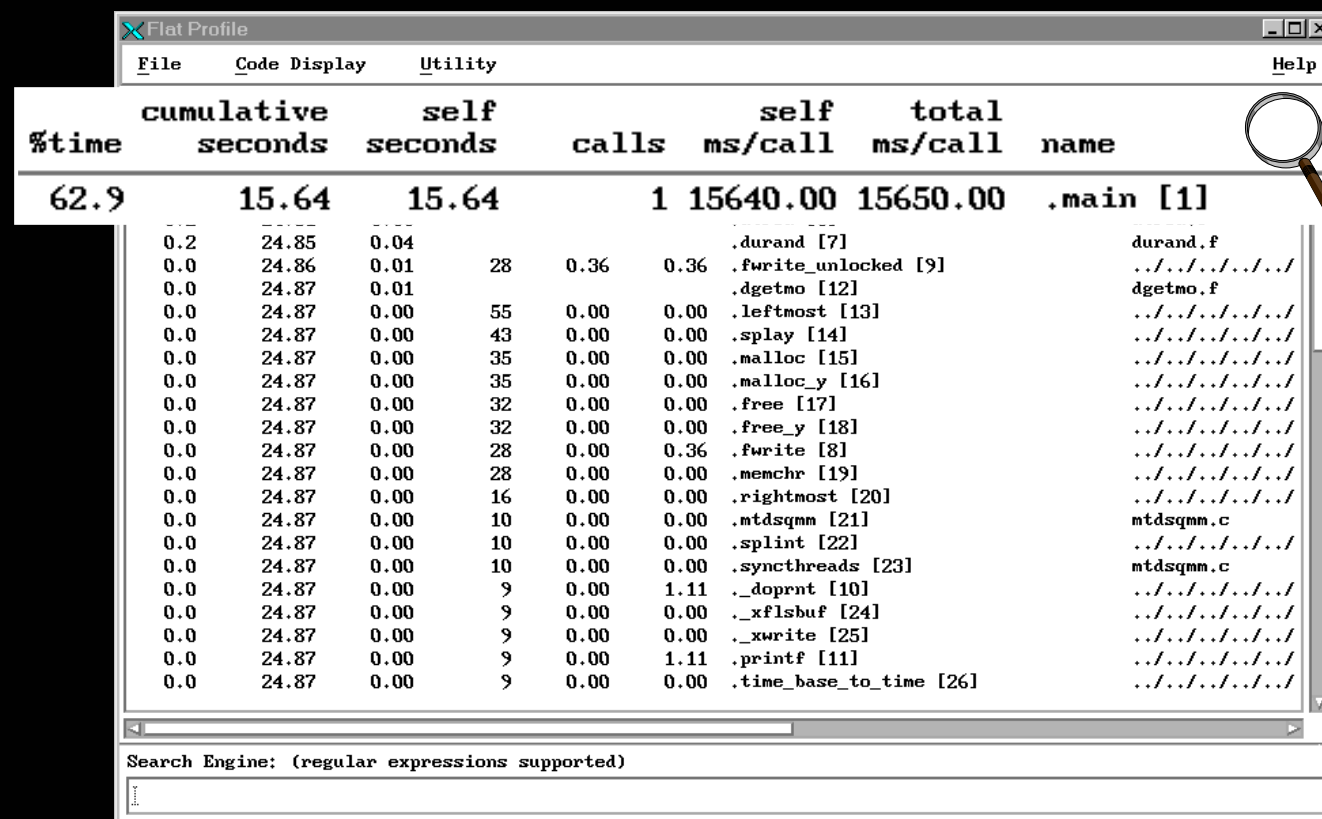
- Flat Profile

- Call Graph Profile

- Function Index

- Function Call Summary

- Library Statistics



%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
62.9	15.64	15.64	1	15640.00	15650.00	.main [1]
0.2	24.85	0.04				.durand [7]
0.0	24.86	0.01	28	0.36	0.36	.fwrite_unlocked [9]
0.0	24.87	0.01				.dgetmo [12]
0.0	24.87	0.00	55	0.00	0.00	.leftmost [13]
0.0	24.87	0.00	43	0.00	0.00	.splay [14]
0.0	24.87	0.00	35	0.00	0.00	.malloc [15]
0.0	24.87	0.00	35	0.00	0.00	.malloc_y [16]
0.0	24.87	0.00	32	0.00	0.00	.free [17]
0.0	24.87	0.00	32	0.00	0.00	.free_y [18]
0.0	24.87	0.00	28	0.00	0.36	.fwrite [8]
0.0	24.87	0.00	28	0.00	0.00	.memchr [19]
0.0	24.87	0.00	16	0.00	0.00	.rightmost [20]
0.0	24.87	0.00	10	0.00	0.00	.mtdsqmm [21]
0.0	24.87	0.00	10	0.00	0.00	.splint [22]
0.0	24.87	0.00	10	0.00	0.00	.syncthread [23]
0.0	24.87	0.00	9	0.00	1.11	._doprnt [10]
0.0	24.87	0.00	9	0.00	0.00	._xflsbuf [24]
0.0	24.87	0.00	9	0.00	0.00	._xwrite [25]
0.0	24.87	0.00	9	0.00	1.11	.printf [11]
0.0	24.87	0.00	9	0.00	0.00	.time_base_to_time [26]

Search Engine: (regular expressions supported)

Xprofiler: Source Code Window

- Source code window displays source code with time profile (in ticks=0.01 sec)
- Access
 - Select function in main display
 - → context menu
 - Select function in flat profile
 - → Code Display
 - → Show Source Code

line	no. ticks per line	source code
202		/*-----*/
203		/* use 2x-unrolling of the outer two loops */
204		/*-----*/
205	4	for (i=i0; i<i0+is-1; i+=2)
206		{
207	8	for (j=j0; j<j0+js-1; j+=2)
208		{
209	1	t11 = c[i*n+j];
210	5	t12 = c[i*n+j+1];
211	5	t21 = c[(i+1)*n+j];
212	19	t22 = c[(i+1)*n+(j+1)];
213		for (k=k0; k<k0+ks; k++)
217	229	t21 = t21 + a[(i+1)*n+k]*bt[j*n+k];
218	144	t22 = t22 + a[(i+1)*n+k]*bt[(j+1)*n+k];
219		}
220	7	c[i*n+j] = t11;
221		c[i*n+j+1] = t12;
222	3	c[(i+1)*n+j] = t21;
223	5	c[(i+1)*n+(j+1)] = t22;
224		}
225		for (j=j; j<j0+js; j++)
226		{
227		t11 = c[i*n+j];
228		t21 = c[(i+1)*n+j];
229		for (k=k0; k<k0+ks; k++)
230		{
231		t11 = t11 + a[i*n+k]*bt[j*n+k];
232		t21 = t21 + a[(i+1)*n+k]*bt[j*n+k];
233		}
234		c[i*n+j] = t11;
235		c[(i+1)*n+j] = t21;
236		}
237		}

Search Engine: (regular expressions supported)

thsub

Xprofiler - Disassembler Code

Disassembler Code for .calc3 [3]					
File		Help			
address	no. ticks per instr.	instruction	assembler code		source code
10002E18	81	FCC4287C	fnms	6, 4, 1, 5	
10002E1C	64	CCF70008	lfd	7, 0x8(23)	POLD(I, J) = P(I, J)+ALPHA*(PNEW(I, J)-
10002E20	187	C90C0008	lfd	8, 0x8(12)	
10002E24	53	C9750008	lfd	11, 0x8(21)	UOLD(I, J) = U(I, J)+ALPHA*(UNEW(I, J)-
10002E28	89	FD63582A	fa	11, 3, 11	
10002E2C	63	FD28387C	fnms	9, 8, 1, 7	POLD(I, J) = P(I, J)+ALPHA*(PNEW(I, J)-
10002E30	4	DD5B0008	stfdu	10, 0x8(27)	U(I, J) = UNEW(I, J)
10002E34		C9540008	lfd	10, 0x8(20)	VOLD(I, J) = V(I, J)+ALPHA*(VNEW(I, J)-
10002E38	113	FCCA302A	fa	6, 10, 6	
10002E3C	27	C8760008	lfd	3, 0x8(22)	POLD(I, J) = P(I, J)+ALPHA*(PNEW(I, J)-
10002E40	87	FD8012FA	fma	12, 0, 11, 2	UOLD(I, J) = U(I, J)+ALPHA*(UNEW(I, J)-
10002E44	35	DCB90008	stfdu	5, 0x8(25)	V(I, J) = VNEW(I, J)
10002E48	4	FC63482A	fa	3, 3, 9	POLD(I, J) = P(I, J)+ALPHA*(PNEW(I, J)-
10002E4C	12	CD5A0008	lfd	10, 0x8(26)	UOLD(I, J) = U(I, J)+ALPHA*(UNEW(I, J)-
10002E50	62	FCC021BA	fma	6, 0, 6, 4	VOLD(I, J) = V(I, J)+ALPHA*(VNEW(I, J)-
10002E54	36	C85B0008	lfd	2, 0x8(27)	UOLD(I, J) = U(I, J)+ALPHA*(UNEW(I, J)-
10002E58	244	DCEC0008	stfdu	7, 0x8(12)	P(I, J) = PNEW(I, J)
10002E5C	28	FD0040FA	fma	8, 0, 3, 8	POLD(I, J) = P(I, J)+ALPHA*(PNEW(I, J)-
10002E60		C8990008	lfd	4, 0x8(25)	VOLD(I, J) = V(I, J)+ALPHA*(VNEW(I, J)-
10002E64	316	DCD40008	stfdu	6, 0x8(20)	
10002E68	29	FC62507C	fnms	3, 2, 1, 10	UOLD(I, J) = U(I, J)+ALPHA*(UNEW(I, J)-

Search Engine: (regular expressions supported)

Xprofiler: Tips and Hints

- Simplest when gmon.out.*, executable, and source code are in one directory
 - Select “**Set File Search Path**” on “**File**” menu to set source directory when source, and executable are not in the same directory
 - Can use **-qfullpath** to encode the path of the source files into the binary
- By default, call tree in main display is “clustered”
 - Menu Filter → Uncluster Functions
 - Menu Filter → Hide All Library Calls
- Libraries must match across systems!
 - on measurement nodes
 - on workstation used for display!
- Must sample realistic problem (sampling rate is 1/100 sec)

HPM

HPM: What Are Performance Counters

- **Extra logic inserted in the processor to count specific events**
- **Updated at every cycle**
- **Strengths:**
 - Non-intrusive
 - Very accurate
 - Low overhead
- **Weakness**
 - Provides only hard counts
 - Specific for each processor
 - Access is not well documented
 - Lack of standard and documentation on what is counted

HPM: Hardware Counters

- **8 counters on PPC970 and Power4, 6 counters on Power 5/5+**
 - Several (100+) events per counter
- **48 UPC counters on Blue Gene/L, 328 events**
- **256 UPC counters on Blue Gene/P, ~1000 events**
- **Events can not be selected independently**
 - PPC970: 41 groups, default: 23
 - Power 4: 64 groups, default: 40
 - Power 5: 140 groups (AIX 5.2), 148 groups (AIX 5.3), default 137
 - Power 5+: 152 groups, default 145
 - Blue Gene/L: 16 groups
- **hpmcount -l or -c for list of counters**

HPM: Hardware Counters

- Cycles
- Instructions
- Floating point instructions
- Integer instructions
- Load/stores
- Cache misses
- TLB misses
- Branch taken / not taken
- Branch mispredictions
- Useful derived metrics
 - **IPC - instructions per cycle**
 - **Float point rate (Mflip/s)**
 - **Computation intensity**
 - **Instructions per load/store**
 - **Load/stores per cache miss**
 - **Cache hit rate**
 - **Loads per load miss**
 - **Stores per store miss**
 - **Loads per TLB miss**
 - **Branches mispredicted %**
- **Derived metrics allow users to correlate the behavior of the application to one or more of the hardware components**
- **One can define threshold values acceptable for metrics and take actions regarding program optimization when values are below the threshold**

HPM Toolkit

- **Data capture, analysis, and presentation of hardware performance metrics for HPC applications and systems**
 - **hpmcount**
 - Starts a program and at the end of the execution provides a summary with hardware counters information and derived metrics
 - Simple to use, no source code modification
 - **libhpm**
 - Instrumentation library for performance measurement of Fortran, C, and C++ applications
 - **hpmstat**
 - Hardware monitoring at system level

HMPCOUNT Usage

- **[poe] hpmcount [-o <name>] [-u] [-n] [-x] [-g <group>] <program>**
- **hpmcount [-h] [-l] [-c]**
 - <program> program to be executed
 - -h displays this help message
 - -o <name> output file name
 - -u make the file name <name> unique
 - -n no hpmcount output in stdout when "-o" flag is used
 - -x adds formulas for derived metrics
 - -g <group> PM_API group number
 - -l list groups
 - -c list counters and events

HPM: Environment Variables

- **HPM_EVENT_SET**
- **HPM_VIZ_OUTPUT**
- **HPM_OUTPUT_NAME**
- **Memory, cache, and TLB miss latencies:**
 - HPM_TLB_LATENCY
 - HPM_MEM_LATENCY
 - HPM_L3_LATENCY
 - HPM_L35_LATENCY
 - HPM_L2_LATENCY
 - HPM_L25_LATENCY
 - HPM_L275_LATENCY
- ...

HPMCOUNT Output - Resource Usage

Total execution time (wall clock time): 25.329646 seconds

Resource Usage Statistics

Total amount of time in user mode	: 21.500000 seconds
Total amount of time in system mode	: 1.820000 seconds
Maximum resident set size	: 25444 Kbytes
Average shared memory use in text segment	: 77688 Kbytes*sec
Average unshared memory use in data segment	: 54386768 Kbytes*sec
Number of page faults without I/O activity	: 6372
Number of page faults with I/O activity	: 8
Number of times process was swapped out	: 0
Number of times file system performed INPUT	: 0
Number of times file system performed OUTPUT	: 0
Number of IPC messages sent	: 0
Number of IPC messages received	: 0
Number of signals delivered	: 1
Number of voluntary context switches	: 168
Number of involuntary context switches	: 2605

HPMCOUNT Output - Group 5

PM_DATA_FROM_L3 (Data loaded from L3)	:	64164220
PM_DATA_FROM_MEM (Data loaded from memory)	:	5623627
PM_DATA_FROM_L35 (Data loaded from L3.5)	:	281896
PM_DATA_FROM_L2 (Data loaded from L2)	:	947542929
PM_DATA_FROM_L25_SHR (Data loaded from L2.5 shared)	:	739
PM_DATA_FROM_L275_SHR (Data loaded from L2.75 shared)	:	567
PM_DATA_FROM_L275_MOD (Data loaded from L2.75 modified)	:	903
PM_DATA_FROM_L25_MOD (Data loaded from L2.5 modified)	:	120

Memory traffic	:	2879.297 MBytes
Memory bandwidth	:	96.317 MBytes/sec
<u>Estimated latency from loads from memory</u>	:	1.730 sec
Total loads from L3	:	64.446 M
L3 traffic	:	8249.103 MBytes
L3 bandwidth	:	275.945 MBytes/sec
<u>Estimated latency from loads from L3</u>	:	5.067 sec
L3 Load miss rate	:	8.026 %
Total loads from L2	:	947.545 M
L2 traffic	:	121285.793 MBytes
L2 bandwidth	:	4057.200 MBytes/sec
<u>Estimated latency from loads from L2</u>	:	8.747 sec
L2 Load miss rate	:	6.886 %

Estimation based
on user input

HPMCOUNT Output - Group 60

PM_FPU_FDIV (FPU executed FDIV instruction)	:	262675952
PM_FPU_FMA (FPU executed multiply-add instruction)	:	4466499631
PM_FPU0_FIN (FPU0 produced a result)	:	6857269712
PM_FPU1_FIN (FPU1 produced a result)	:	9200414352
PM_CYC (Processor cycles)	:	152475196089
PM_FPU_STF (FPU executed store instruction)	:	3430135817
PM_INST_CMPL (Instructions completed)	:	117862217347
PM_LSU_LDF (LSU executed Floating Point load instruction)	:	17921080598

Measurement of
system activity:

User time / WCT

Utilization rate	:	99.658 %
Load and store operations	:	21351.216 M
Instructions per load/store	:	5.520
MIPS	:	1001.455
Instructions per cycle	:	0.773
HW Float point instructions per Cycle	:	0.105
Floating point instructions + FMAs (flips)	:	17094.048 M
Flip rate (flips / WCT)	:	145.245 Mflip/sec
Flips / user time	:	145.743 Mflip/sec
Weighted Floating Point Instructions	:	20771.511 M
Weighted flip rate	:	176.492 M Wflip/sec
FMA percentage	:	52.258 %
Computation intensity	:	0.801

Measurement of
FPU & LSU utilization:
Flips / I.S

Derived Metrics

- Utilization rate
- Total FP load and store operations
- MIPS
- Instructions per cycle/run cycle/load store
- % Instructions dispatched that completed
- Fixed point operations per Cycle or load/stores
- Branches mispredicted percentage
- number of loads per load miss
- number of stores per store miss
- number of load/stores per L1 miss
- L1 cache hit rate
- number of loads per TLB miss
- number of loads/stores per TLB miss
- Total Loads from L2
- L2 load traffic
- L2 load bandwidth per processor
- Estimated latency from loads from L2
- % loads from L2 per cycle
- Total Loads from local L2
- local L2 load traffic
- local L2 load bandwidth per processor
- Estimated latency from loads from local L2
- % loads from local L2 per cycle
- Total Loads from L3
- L3 load traffic
- L3 load bandwidth per processor
- Estimated latency from loads from L3
- ...

LIBHPM

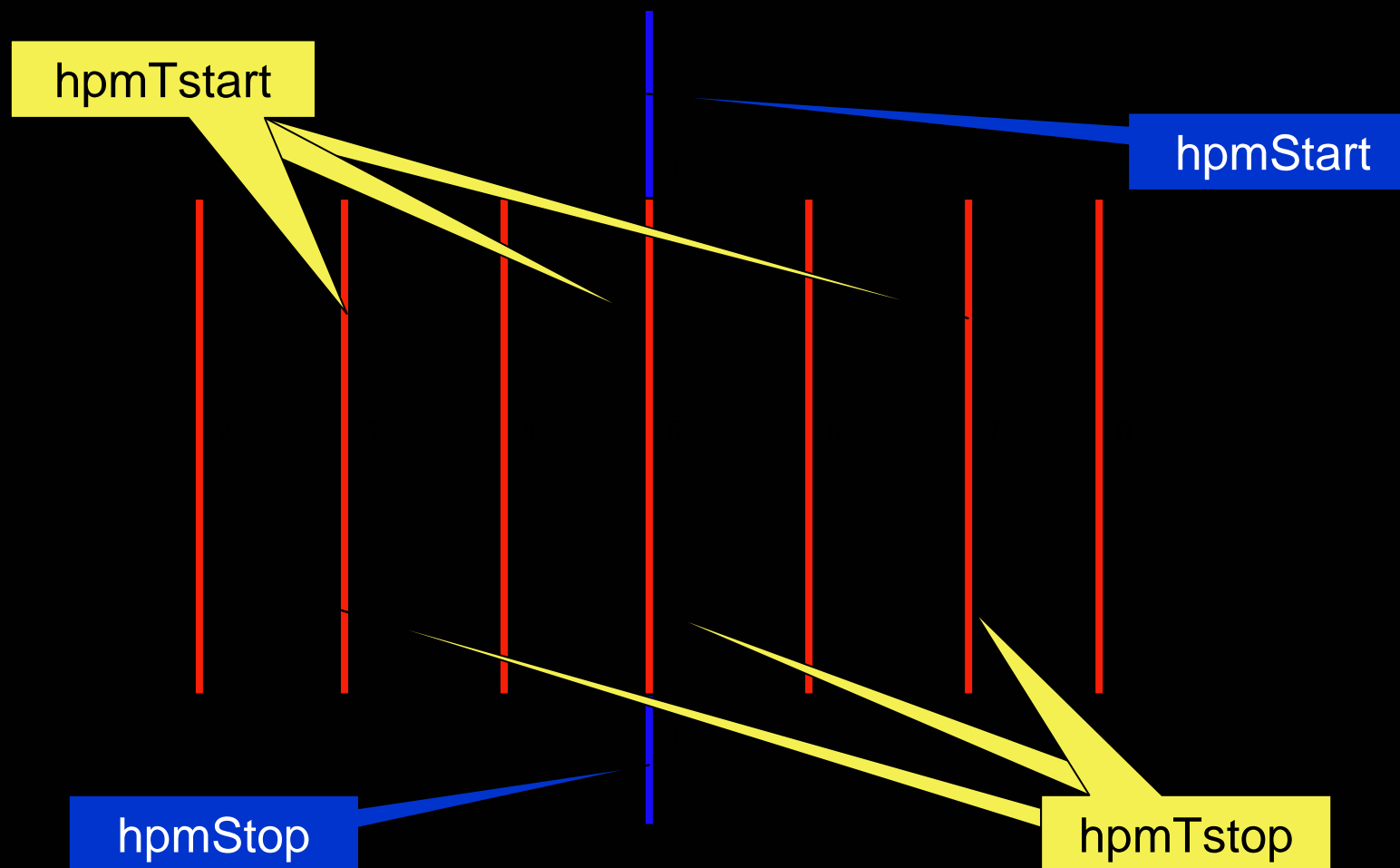
- **Allows to go in the source code and instrument different sections independently**
- **Supports Fortran, C, and C++**
- **For each instrumented section provides:**
 - Total count & duration (wall clock time)
 - Hardware performance counters information
 - Derived metrics
- **Provides resource usage statistics for the total execution of the instrumented program**
- **Supports:**
 - MPI, OpenMP, & pThreads
 - Multiple instrumentation points
 - Nested instrumentation
 - Multiple calls to an instrumented point

Using LIBHPM

- **Declaration:**
- **Use:**
 - #include f_hpm.h
 - call **f_hpm**init(0, "prog")
 - call **f_hpm**start(1, "work")
 - do
 - call do_work()
 - call **f_hpm**start(22, "more work")
 - call compute_meaning_of_life()
 - call **f_hpm**stop(22)
 - end do
 - call **f_hpm**stop(1)
 - call **f_hpm**terminate(0)

Use MPI
taskID with
MPI programs

HPM: Multi-thread Support



hpmviz

File

swim_omp

swim_omp.f

calc1.f

calc2.f

calc3.f

Label	ExcSec	IncSec	Count
Loop 300	4.572	4.572	2398
Loop 200	4.203	4.203	2400
Loop 100	3.071	3.071	2400
Calc3	1.838	6.813	2398

```

*   VOLD(N1,N2), POLD(N1,N2),
2   CU(N1,N2), CV(N1,N2),
*   Z(N1,N2), H(N1,N2), PSI(N1,N2)
C
COMMON /CONS/ DT,TD,DX,DY,A,ALPHA,ITMAX,MPRINT
1   NP1,EL,PI,TPI,DI,DJ,PCF

```

Metric Option:

- ☐ Count
- ☐ ExcSec
- ☐ IncSec
- ☐ PM_FPU_FDIV
- ☐ PM_FPU_FMA
- ☐ PM_FPU0_FIN
- ☐ PM_FPU1_FIN
- ☐ PM_CYC
- ☐ PM_FPU_STF
- ☐ PM_INST_CMPL
- ☐ PM_LSU_LDF
- ☐ U time
- ☐ Use rate
- ☐ (M) LS
- ☐ MIPS
- ☐ HW FP/Cyc
- ☐ Instr/LS
- ☐ M Flips
- ☐ IpC
- ☐ Mflip/s
- ☐ WFlips
- ☐ Wflip/s
- ☐ FMA %
- ☐ Comp Int.

Metric Browser: Loop 300

Close		Metric Options		Precision														
Node	Thread	Count	ExcSec	IncSec	U time	Use rate	(M) LS	MIPS	HW FP/Cyc	Instr/LS	M Flips	IpC	Mflip/s	WFlips	Wf			
MPI	0	3	2398	4.539	4.539	3.923	86.425	590.056	291.855	0.116	2.245	589.86	0.26	129.947	589.86	12		
MPI	0	0	2398	4.572	4.572	4.378	95.763	608.414	263.277	0.107	1.978	608.234	0.211	133.037	608.234	13		
MPI	0	2	2398	4.549	4.549	4.366	95.979	590.019	255.241	0.104	1.968	589.838	0.205	129.663	589.838	12		
loop	0	1	2398	4.547	4.547	4.308	94.759	590.024	259.19	0.105	1.997	589.837	0.21	129.728	589.837	12		
Calc	1	2	2398	4.534	4.534	4.398	96.999	590.044	253.123	0.103	1.945	589.856	0.201	130.088	589.856	13		
ALL	1	1	2398	4.528	4.528	3.942	87.069	589.983	286.058	0.115	2.195	589.807	0.253	130.263	589.807	13		
Initial	1	0	2398	4.547	4.547	3.766	82.828	608.434	308.065	0.124	2.302	608.244	0.286	133.762	608.244	13		
Calc	2	3	2398	4.523	4.523	3.537	78.198	589.962	317.346	0.128	2.433	589.781	0.312	130.4	589.781	13		
	2	0	2398	4.538	4.538	3.777	83.218	608.448	312.01	0.124	2.327	608.262	0.288	134.029	608.262	13		
	2	2	2398	4.522	4.522	4.313	95.364	590.033	257.962	0.105	1.977	589.86	0.208	130.431	589.86	13		
	2	3	2398	4.52	4.52	4.307	95.285	589.985	258.863	0.105	1.983	589.806	0.209	130.492	589.806	13		
	2	1	2398	4.52	4.52	4.35	96.222	589.943	255.814	0.104	1.96	589.767	0.205	130.466	589.767	13		
	3	3	2398	4.487	4.487	4.193	93.453	571.551	259.827	0.105	2.04	571.374	0.214	127.352	571.374	12		
	3	1	2398	4.502	4.502	4.365	96.953	589.937	254.196	0.104	1.94	589.763	0.202	131.003	589.763	13		
	3	2	2398	4.483	4.483	4.139	92.33	571.556	263.864	0.106	2.07	571.38	0.22	127.445	571.38	12		
	3	0	2398	4.506	4.506	3.927	87.154	590.044	290.852	0.116	2.221	589.856	0.257	130.901	589.856	13		

```

VOLD(I,J) = V(I,J)+ALPHA*(VNEW(I,J)-2.*V(I,J)+VOLD(I,J))
POLD(I,J) = P(I,J)+ALPHA*(PNEW(I,J)-2.*P(I,J)+POLD(I,J))
U(I,J) = UNEW(I,J)
V(I,J) = VNEW(I,J)
P(I,J) = PNEW(I,J)
300 CONTINUE
call f_hpmstop( 30+omp_get_thread_num())

```

MPI Profiler/Tracer

Message-Passing Performance

MP_Profiler

- Implements wrappers around MPI calls using the PMPI interface
 - start timer
 - call pmpi equivalent function
 - stop timer
- Captures “summary” data for MPI calls with source code traceback
- No changes to source code, but MUST compile with -g
- ~1.7 microsecond overhead per MPI call
- Does not synchronize MPI calls
- Compile with -g and link with libmpitrace.a
- Generate XML files for peekperf

Message-Passing Performance

MPI Tracer

- Captures “timestamped” data for MPI calls with source traceback
- Provides a color-coded trace of execution
- ~1.4 microsecond overhead per MPI call
- Very useful to identify load-balancing issues

PeekPerf Main Window

File Tools Options

GAMESS MPI_Application

Label ▾	Call Count [Ma
---MPI_Allreduce_807	16
---MPI_Allreduce_868	38
---MPI_Barrier_1496	1
---MPI_Barrier_248	1
---MPI_Barrier_765	4
---MPI_Bcast_924	285
---MPI_Comm_rank_973	5
---MPI_Comm_size_972	5
---MPI_Iprobe_1160	68
---MPI_Recv_1027	1
---MPI_Recv_1135	2
---MPI_Ssend_1002	2
Summary_MPI_Al	
Summary_MPI_Ba	
Summary_MPI_Bc	
Summary_MPI_Co	
Summary_MPI_Co	
Summary_MPI_Ip	
Summary_MPI_Re	
Summary_MPI_Ss	

gamess.f ddi.f

```

OF COMPUTE
C PROCESSES ONLY, NOT THE TOTAL NUMBER OF
PROCESSES.
C
-----
C
      IMPLICIT NONE
      INTEGER DDI_NP, DDI_ME

C
      INCLUDE 'mpif.h'
      INTEGER*4 DDI_NP4,DDI_ME4,IERROR

C
      CALL MPI_COMM_SIZE( MPI_COMM_WORLD, DDI_NP4,
IERROR )

      CALL MPI_COMM_RANK( MPI_COMM_WORLD, DDI_ME4,

```

Metric Browser: MPI_Bcast_924

Close	Metric Options ▾	Precision ▾					
Task ▾	Message Size	WallClock	Count	Call Count [Max]	WallClock [Max]	Transferred Bytes	
0	(2) 5 ... 16	0.049632	133	133	0.049632	1064	
0	(3) 17 ... 64	0.000144	2	2	0.000144	64	
0	(4) 65 ... 256	0.001124	16	16	0.001124	1408	
0	(5) 257 ... 1K	0.030647	163	163	0.030647	47408	
0	(6) 1K ... 4K	0.011084	134	134	0.011084	198472	
0	(7) 4K ... 16K	0.024244	285	285	0.024244	1.69084e+06	
0	(8) 16K ... 64K	0.001328	16	16	0.001328	227200	
0	(9) 64K ... 256K	0.078051	121	121	0.078051	1.09938e+07	
1	(2) 5 ... 16	0.185036	133	133	0.185036	1064	
1	(3) 17 ... 64	0.000183	2	2	0.000183	64	
1	(4) 65 ... 256	0.009773	16	16	0.009773	1408	
1	(5) 257 ... 1K	0.018897	163	163	0.018897	47408	
1	(6) 1K ... 4K	0.019423	134	134	0.019423	198472	
1	(7) 4K ... 16K	0.251916	285	285	0.251916	1.69084e+06	
1	(8) 16K ... 64K	0.406296	16	16	0.406296	227200	
1	(9) 64K ... 256K	0.087307	121	121	0.087307	1.09938e+07	

SHMEM Profiling Capability

PeekPerf Main Window

File Tools Options

SHMEM_Application

Label	Call Count
my_pe_945	2
num_pes_944	2
shmem_barrier_all_788	431
shmem_barrier_all_799	431
shmem_barrier_all_859	186
shmem_barrier_all_871	186
shmem_barrier_all_915	114
shmem_barrier_all_927	114
shmem_broadcast8_796	273
shmem_int8_sum_to_all_923	114

ddishm.f

```

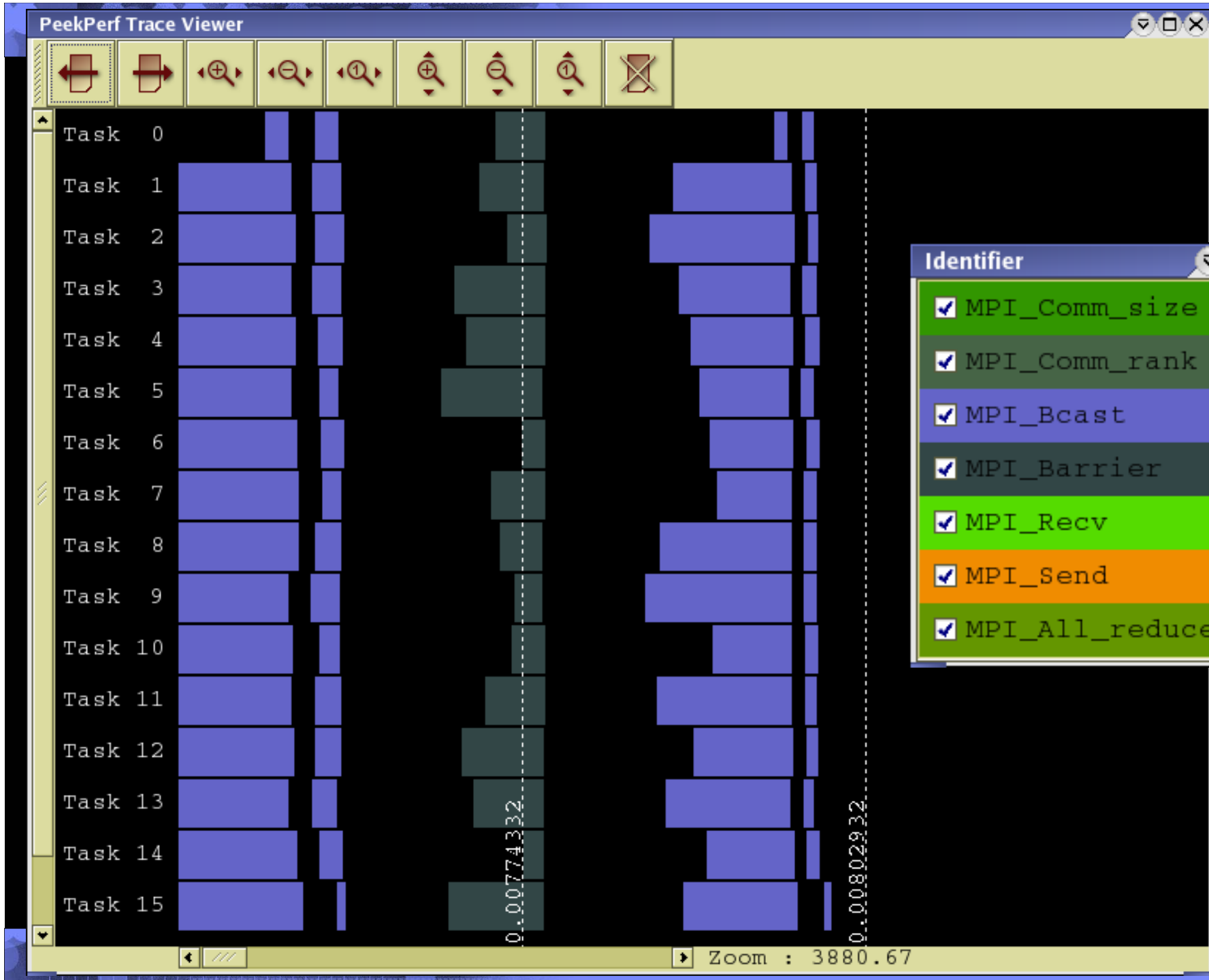
WPE_START = PE_START
WLOG_STRIDE = LOG_STRIDE
WPE_SIZE = PE_SIZE
CALL SHMEM_BROADCAST8( TARGET, SOURCE, WLENGTH
* WPE_START, WLOG_STRIDE, W
CALL SHMEM_QUIET()
CALL SHMEM_BARRIER_ALL()
IF ( MYNODE .NE. PE_ROOT )
* CALL ICOPY( LENGTH, TARGET, 1, BUFF(LOCBUF), 1
LOCBUF = LOCBUF + LENGTH
END DO

```

Metric Browser: shmem_broadcast8_796

Close Metric Options Precision

Task	Message Size	Call Count [Max]	WallClock [Max]	Transferred Bytes	Count	WallClock
0	(2) 5 ... 16	49	0.040212	392	49	0.040212
0	(3) 17 ... 64	7	0.002289	168	7	0.002289
0	(4) 65 ... 256	48	0.015485	3192	48	0.015485
0	(5) 257 ... 1K	273	0.105899	76440	273	0.105899
0	(6) 1K ... 4K	29	0.019601	20888	29	0.019601
0	(7) 4K ... 16K	25	0.008123	162816	25	0.008123
1	(2) 5 ... 16	49	0.020304	392	49	0.020304
1	(3) 17 ... 64	7	0.003646	168	7	0.003646
1	(4) 65 ... 256	48	0.029487	3192	48	0.029487
1	(5) 257 ... 1K	273	0.142042	76440	273	0.142042
1	(6) 1K ... 4K	29	0.184179	20888	29	0.184179
1	(7) 4K ... 16K	25	0.015682	162816	25	0.015682



Identifier

- ☒ MPI_Comm_size
- ☒ MPI_Comm_rank
- ☒ MPI_Bcast
- ☒ MPI_Barrier
- ☒ MPI_Recv
- ☒ MPI_Send
- ☒ MPI_All_reduce

PompProf

“Standard” OpenMP Monitoring API?

- **Problem:**

- OpenMP (unlike MPI) does not define standard monitoring interface (at SC06 they accepted a proposal from SUN and others)
- OpenMP is defined mainly by directives/pragmas

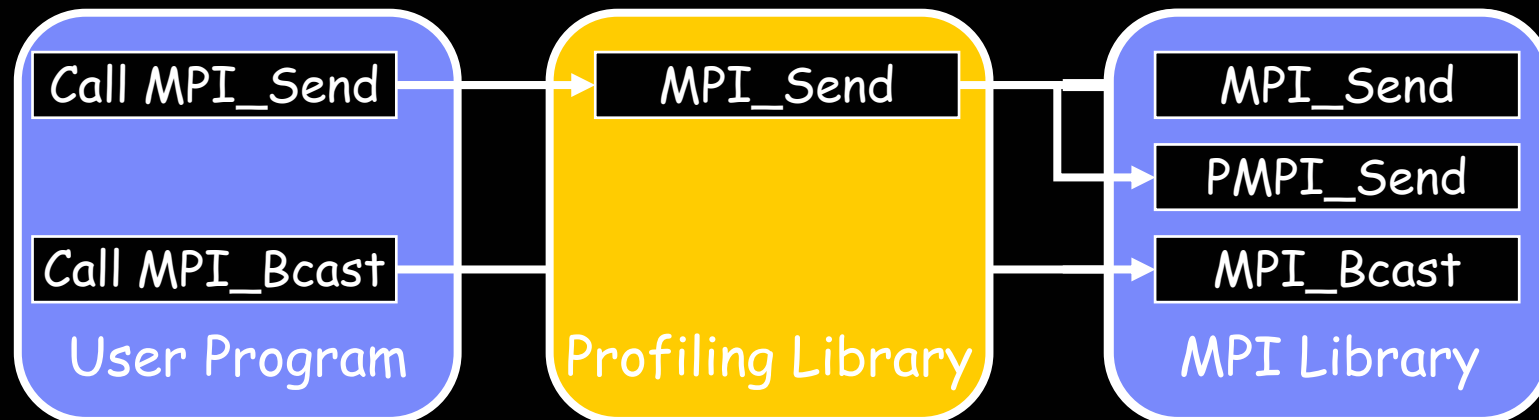
- **Solution:**

- **POMP**: OpenMP Monitoring Interface
- Joint Development
 - Forschungszentrum Jülich
 - University of Oregon
- Presented at EWOMP'01, LACSI'01 and SC'01
 - “The Journal of Supercomputing”, 23, Aug. 2002.



Profiling of OpenMP Applications: POMP

- Portable cross-platform/cross-language API to simplify the design and implementation of OpenMP tools
- POMP was motivated by the MPI profiling interface (PMPI)
 - **PMPI allows selective replacement of MPI routines at link time**
 - **Used by most MPI performance tools (including MPI Profiler/Tracer)**



POMP Proposal

- **Three groups of events**
 - **OpenMP constructs and directives/pragmas**
 - Enter/Exit around each OpenMP construct
 - Begin/End around associated body
 - Special case for parallel loops:
 - ChunkBegin/End, IterBegin/End, or IterEvent instead of Begin/End
 - “Single” events for small constructs like atomic or flush
 - **OpenMP API calls**
 - Enter/Exit events around `omp_set_*_lock()` functions
 - “single” events for all API functions
 - **User functions and regions**
- **Allows application programmers to specify and control amount of instrumentation**

Example: POMP Instrumentation

```
1:  int main() {
2:      int id;
***   POMP_Init();
3:
***   { POMP_handle_t pomp_hdl = 0;
***       int32 pomp_tid = omp_get_thread_num();
***       POMP_Parallel_enter(&pomp_hdl, pomp_tid, -1, 1,
***           "49*type=pregion*file=demo.c*slines=4,4*elines=8,8**");
4:   #pragma omp parallel private(id)
5:   {
***       int32 pomp_tid = omp_get_thread_num();
***       POMP_Parallel_begin(pomp_hdl, pomp_tid);
6:       id = omp_get_thread_num();
7:       printf("hello from %d\n", id);
***       POMP_Parallel_end(pomp_hdl, pomp_tid);
8:   }
***       POMP_Parallel_exit(pomp_hdl, pomp_tid);
***   }
***   POMP_Finalize();
9: }
```


DPOMP: Dynamic Performance Monitoring Interface for OpenMP

- **Collaboration with Forschungszentrum Jülich**
- **Motivation:**
 - POMP was under review (at the time)
 - May take too long to be implemented (if accepted)
- **Approach**
 - A POMP implementation based on dynamic probes
 - Built on top of DPCL
 - Modifies the binary with performance instrumentation
 - No source code or re-compilation required

SIGMA-POMP: Performance Monitoring Interface for OpenMP based on PSIGMA Instrumentation

- **Approach**

- A POMP implementation using pSigma's binary instrumentation and rewriting
- Built on top of pSigma
 - Modifies the binary with performance instrumentation
 - No source code or re-compilation required

POMP Profiler (PompProf)

- **Profiler for OpenMP application implemented on top of DPOMP and SIGMA-POMP**
- **Generates a detailed profile describing overheads and time spent by each thread in three key regions of the parallel application:**
 - Parallel regions
 - OpenMP loops inside a parallel region
 - User defined functions
 -
- **Profile data is presented in the form of an XML file that can be visualized with PeekPerf**

peekperf

File Tools

main.f

Label	Count	Excl. Time	Incl. Time	%Total Overhead	%Imbalance	A
pregon_324	1	54.4638	128.508	4.1e-05	6.3e-05	12
loop_1125	10	13.3219	13.3219	0.005431	88.4638	21
loop_852	10	12.854	12.854	0.005178	52.5563	17
loop_549	10	12.3907	12.3907	0.005676	98.8048	21
loop_1685	10	12.3036	12.3036	0.005682	62.1319	16
loop_1408	10	11.8975	11.8975	0.005578	96.0548	19
loop_1934	10	10.1843	10.1843	0.006926	41.4999	13
loop_790	1	0.522151	0.522151	0.013839	37.576	0.4
loop_1668	1	0.485633	0.485633	0.00896	36.8549	0.4
loop_1623	1	0.039318	0.039318	0.596515	2.79627	0.0
loop_1379	1	0.031769	0.031769	0.198128	13.846	0.0
func_rdparam_174	1	0.01605	0.01605	0	0	0
loop_855	1	0.013306	0.013306	0.224878	342.237	0.0
func_main_155	1	0.000771	128.525	0	0	0
func_deltat_550	1	2.2e-05	2.2e-05	0	0	0
func_layout_296	1	6e-06	6e-06	0	0	0
func_changedir_1841	1	3e-06	3e-06	0	0	0

main.f runhyd3.f

```

& msg_mxzbdy, msg_mxxbdy, msg_mxybdy)
endif

c$omp do schedule(static)
do ip=1,nchunk

if ((ithread.eq.1).and.(ip.ge.icomm_point(1,4)).and.
& (irec1br.eq.0) .and. (iflag.eq.0)) then
irec1br = 1

call bdrys1br(dddo, nz, nx, ny, 5,
& msg_zm, msg_zp, msg_xm, msg_xp, msg_ym, msg_yp,
& msg_mnzbdy, msg_mnxbdy, msg_mnybdy,
& msg_mxzbdy, msg_mxxbdy, msg_mxybdy)
call bdry2o1s(dddo, nz, nx, ny, 5,
& msg_zm, msg_zp, msg_xm, msg_xp, msg_ym, msg_yp,
& msg_mnzbdy,
& msg_mnybdy,
& msg_mxybdy)
xp, msg_ym, msg_yp,
& msg_mnzbdy, msg_mnxbdy, msg_mnybdy,
& msg_mxzbdy, msg_mxxbdy, msg_mxybdy)
call bdry3o1r(dddo, nz, nx, ny, 5,

```

Metric Browser: loop_1125

Close Metric Options Precision

Task	thread	Time in Master	TT: Thread Time	CT: Computation Time	%Imbalance	TO = TT - CT	%TO (Barrier)	%TO (RTL)
0	0	13.3219	13.3219	13.3211	0	0.000723	0.000275	0.005156
0	1	0	15.8615	15.861	19.0664	0.000504	0.000108	0.003679
0	2	0	21.0295	21.029	57.8616	0.000514	0.000103	0.003753
0	3	0	24.4342	24.4338	83.4208	0.000416	4e-06	0.003119
0	4	0	24.1806	24.1802	81.5175	0.000423	0	0.003173
0	5	0	25.0957	25.0952	88.3864	0.00047	1.4e-05	0.00351
0	6	0	25.1062	25.1055	88.4638	0.00062	0.000157	0.004495
0	7	0	23.9859	23.9854	80.0548	0.000556	0.000112	0.004063

SIGMA

SIGMA Infrastructure: Highlights

Complex infrastructure: symbolic, dynamic, performance-oriented environment for binary instrumentation

- Inject arbitrary user-supplied probes into a binary application

Provides a platform to:

- Develop performance tools
- Experiment with memory performance models
- Ask “what-if” questions regarding data and code rearrangements
- Provide feedback on design of new memory architectures
- Identify performance bottlenecks due to the memory hierarchy and data layout

SIGMA Memory Profiler is integrated in the HPC Toolkit nm

Memory Profile:

Provide counters such as hits, misses, cold misses for

- each cache level
- each function
- each data structure
- each data structure within each function

Output sorted by the SIGMA **memtime**:

- $\text{SUM}(\text{LoadHits}(i) * \text{LoadLat}(i) + \text{StoreHits}(i) * \text{StoreLat}(i)) +$
- $\text{\#TLBmisses} * \text{Lat}(\text{TLBmiss})$
- **memtime should track wall time for memory bound applications**

Memory Profile Output

	L1	L2	L3	TLB	MEM
FUNCTION: calc1 (memtime = 0.0050)					
Load Acc/Miss/Cold	522819/2252/1	2252/345/0	345/0/0	784419/126/0	0/-/-
Load Acc/Miss Ratio	232	6	-	6225	-
Load Hit Ratio	99.57%	84.68%	100.00%	99.98%	-
Est. Load Latency	0.0008 sec	0.0000 sec	0.0000 sec	0.0001 sec	0.0000 sec
Load Traffic	-	238.38 Kb	43.12 Kb	-	0.00 Kb
.....					

FUNCTION: calc2 (memtime = 0.0042)					
Load Acc/Miss/Cold	622230/2631/0	2631/1661/0	1661/0/0	814269/94/0	0/-/-
Load Acc/Miss Ratio	236	1	-	8662	-
Load Hit Ratio	99.58%	36.87%	100.00%	99.99%	-
Est. Load Latency	0.0010 sec	0.0000 sec	0.0001 sec	0.0001 sec	0.0000 sec
Load Traffic	-	121.25 Kb	207.62 Kb	-	0.00 Kb
.....					

	L1	L2	L3	TLB	MEM
DATA: u (memtime = 0.0012)					
Load Acc/Miss/Cold	167710/708/0	708/317/0	317/0/0	216097/31/0	0/-/-
Load Acc/Miss Ratio	236	2	-	6970	-
Load Hit Ratio	99.58%	55.23%	100.00%	99.99%	-
Est. Load Latency	0.0003 sec	0.0000 sec	0.0000 sec	0.0000 sec	0.0000 sec
Load Traffic	-	48.88 Kb	39.62 Kb	-	0.00 Kb
.....					

DATA: v (memtime = 0.0012)					
Load Acc/Miss/Cold	167710/721/0	721/316/0	316/0/0	216097/31/0	0/-/-
Load Acc/Miss Ratio	232	2	-	6970	-
Load Hit Ratio	99.57%	56.17%	100.00%	99.99%	-
Est. Load Latency	0.0003 sec	0.0000 sec	0.0000 sec	0.0000 sec	0.0000 sec
Load Traffic	-	50.62 Kb	39.50 Kb	-	0.00 Kb
.....					

Memory Profile Viewer – Data Structure Focus

peekperf

File Tools

sigma

Label	MemTime /	Access
u@Global	1.57702e+06	216097
u@Global_inital	447628	32511
u@Global_calc3	444718	32258
u@Global_calc3z	433822	32258
u@Global_calc1	250850	119070
v@Global	1.57569e+06	216097
p@Global	1.52905e+06	192786
h@Global	1.30777e+06	168216
z@Global	1.30759e+06	168216
pold@Global	1.27448e+06	112144
vold@Global	1.27217e+06	112144
uold@Global	1.27047e+06	112144
cv@Global	1.26258e+06	145158
cu@Global	1.26066e+06	145158
unew@Global	1.19097e+06	81151
vnew@Global	1.15966e+06	81151
pnew@Global	1.15684e+06	81151
psi@Global	486142	51913
unknownDt@Global	11220	539
unknownDt@Local	424	30

swim.f

```

VOID(N1,N2), FOLD(N1,N2),
2  CU(N1,N2), CV(N1,N2),
*  Z(N1,N2), H(N1,N2), PSI(N1,N2)
C
COMMON /CONS/ DT, TDT, DX, DY, A, ALPHA, ITMAX, MPRINT, M, N, MP1,
1  NP1, EL, PI, TPI, DI, DJ, PCF
C
TDT = TDT+TDT
DO 400 J=1, NP1
DO 400 I=1, MP1
UOLD(I,J) = U(I,J)
VOLD(I,J) = V(I,J)
POLD(I,J) = P(I,J)
U(I,J) = UNEW(I,J)
V(I,J) = VNEW(I,J)
P(I,J) = PNEW(I,J)
400 CONTINUE
RETURN
END
C SPEC removed CCMIC$ MICRO
SUBROUTINE CALC3
C
C  TIME SMOOTHER
C
IMPLICIT REAL*8      (A-H, O-Z)
#include "swim.h"
COMMON U(N1,N2), V(N1,N2), P(N1,N2)

```

Metric Browser: u@Global_calc3

Close Metric Options Precision

Task	thread	Misses	LoadMisses	StoreMisses	Hits	LoadHits	StoreHits	Access	LoadAccesses	StoreAccess
0	L1	442	193	249	31816	15936	15880	32258	16129	16129
0	L2	190	74	116	16132	119	16013	16322	193	16129
0	L3	0	0	0	1191	74	1117	1191	74	1117
0	TLB	0	0	0	32258	32258	0	32258	32258	0

C SPEC removed CCMIC\$ DO GLOBAL

sigma

Label	MemTime /	Access
calc1	4.91489e+06	530799
calc1_cv@Global	1.05255e+06	49146
calc1_h@Global	1.0522e+06	49146
calc1_z@Global	1.05208e+06	49146
calc1_cu@Global	1.05142e+06	49146
calc1_u@Global	250850	119070
calc1_v@Global	250270	119070
calc1_p@Global	204374	96012
calc1_unknownDt@Global	1150	63
calc2	4.55976e+06	720579
inital	3.0725e+06	197881
inital_psi@Global	486142	51913
inital_u@Global	447628	32511
inital_v@Global	447504	32511
inital_p@Global	447502	32258
inital_pold@Global	414604	16129
inital_uold@Global	413518	16129
inital_vold@Global	412818	16129
inital_unknownDt@Global	2758	299
inital_unknownDt@Local	26	2
calc3	2.73913e+06	241205
calc3z	2.35196e+06	193568
shalow	4496	63

swim.f

```

DO 86 I=1,MP1
  UOLD(I,J) = U(I,J)
  VOLD(I,J) = V(I,J)
  POLD(I,J) = P(I,J)
86 CONTINUE
C   END OF INITIALIZATION
  RETURN
  END
C SPEC removed CCMIC$ MICRO
  SUBROUTINE CALC1
C
C   COMPUTE CAPITAL U, CAPITAL V, Z AND H
C
  IMPLICIT REAL*8      (A-H, O-Z)
#include "swim.h"

  COMMON U(N1,N2), V(N1,N2), P(N1,N2),
*   UNEW(N1,N2), VNEW(N1,N2),
1   PNEW(N1,N2), UOLD(N1,N2),
*   VOLD(N1,N2), POLD(N1,N2),
2   CU(N1,N2), CV(N1,N2),
*   Z(N1,N2), H(N1,N2), PSI(N1,N2)
C
  COMMON /CONS/ DT, TDT, DX, DY, A, ALPHA, ITMAX, MPRINT, M, N, MP1,
1   NP1, EL, PL, TPL, DLD, J, PCE

```

Metric Browser: calc1_cv@Global

Close

Metric Options

Precision

sses	StoreAccesses	LoadMissesCold	StoreMissesCold	MemTime	ConflictPressure	CapacityPressure	ColdPressure
48387	0	0	1009	1.05255e+06	0.750824	99.2492	1.00757
48387	0	0	1009	0	93.7942	6.20579	16.114
3027	0	0	253	0	91.6419	8.35811	11.9644
0	32	0	0	0	99.9349	0.065112	1535.81

```

1   -U(I+1,J))/ (P(I,J)+P(I+1,J)+P(I+1,J+1)+P(I,J+1))
  H(I,J) = P(I,J)+.25D0*(U(I+1,J)*U(I+1,J)+U(I,J)*U(I,J)
1   +V(I,J+1)*V(I,J+1)+V(I,J)*V(I,J))
100 CONTINUE

```

Usage:

- Binary instrumentation:
 - `sigmaInst -d appbin` (produces executable `appbin.inst`)
 - `./appbin.inst [app-args] -sigma sigmaArchFile`

Things to be aware of:

- It's a simulator!! (may not want to use on long-running apps)
- Overhead is about 2 to 3 orders of magnitude slower
- Only functions statically linked in the executable can be instrumented
- Application must be compiled with `-g`

How to overcome the speed limitations of the simulator? Partial Instrumentation

– Statically select functions to instrument:

- `signalInst -d -dfunc f1,f2,...,fn appbin`

– Dynamically select code sections to instrument:

- `#include "signal_sigma.h"`
- `for (lp=1; lp<NIT; lp++) {`
- `/* start sigma */`
- `if ((lp == 2) || (lp==7)) {`
- `signal_sigma TRACE_ON, lp;`
- `}`
- `for (i=0; i<n; i++) {`
- `for (k=0; k< n; k++) {`
- `u[n * i + k] = 0.0;`
- `f[n * i + k] = rhs;`
- `}`
- `}`
- `/* stop sigma */`
- `if ((lp == 2) || (lp==7)) {`
- `signal_sigma TRACE_OFF, lp;`
- `}`
- `}`

Use partial instrumentation with a grain of salt!

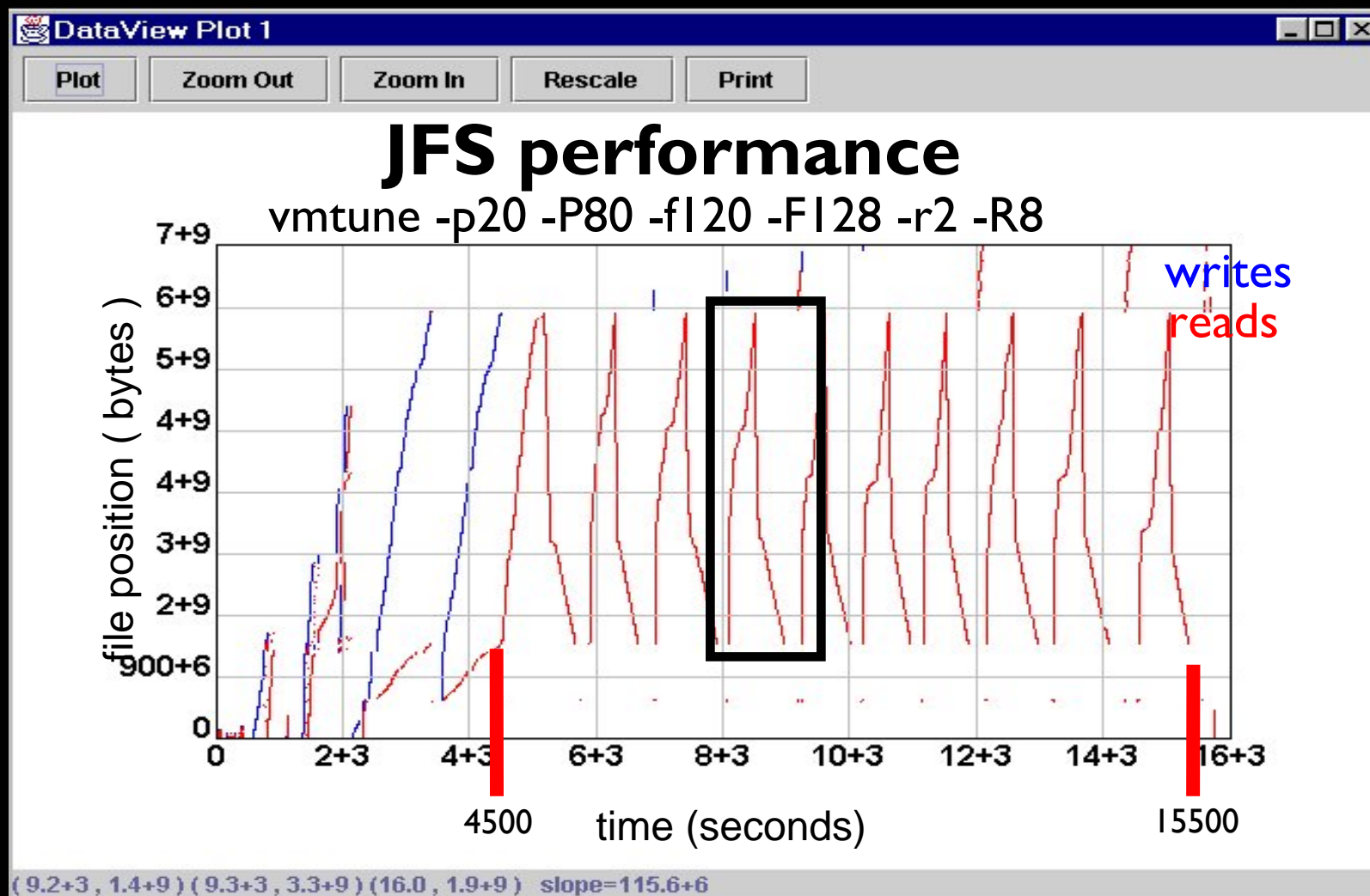
- Unless the entire application is instrumented, there will be **approximation** in the results (cache state won't be accurate between activation/deactivation)
- Use dynamic selection at **phase or loop boundaries** or when the cache can be assumed 'cold'
- Use **sigma signals** to reset the cache
- Ongoing research projects to find **optimal sampling techniques** (collaboration with UMD, SDSC)
- Ongoing research on **automatic sampling**

MIO

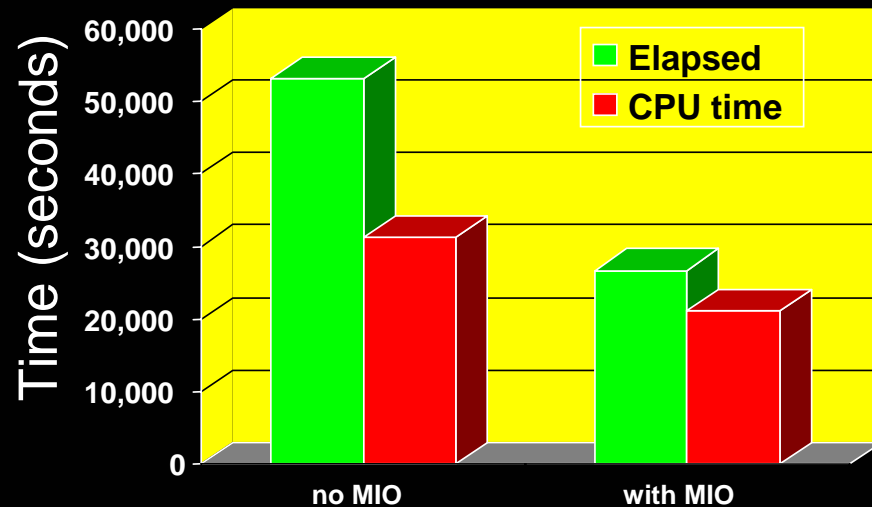
Modular I/O Performance Tool (MIO)

- **I/O Analysis**
 - Trace module
 - Summary of File I/O Activity + Binary Events File
 - Low CPU overhead
- **I/O Performance Enhancement Library**
 - Prefetch module (optimizes asynchronous prefetch and write-behind)
 - System Buffer Bypass capability
 - User controlled pages (size and number)
- **Recoverable Error Handling**
 - Recover module (monitors return values and error + reissues failed requests)
- **Remote Data Server**
 - Remote module (simple socket protocol for moving data)
- **Shared object library for AIX**

Performance Visualization (work in progress)



MSC.Nastran V2001



Benchmark:

SOL 111, 1.7M DOF, 1578 modes, 146 frequencies, residual flexibility and acoustics. 120 GB of disk space.

Machine:

4-way, 1.3 GHz p655, 32 GB with 16 GB large pages, JFS striped on 16 SCSI disks.

MSC.Nastran:

V2001.0.9 with large pages,
dmp=2 parallel=2 mem=700mb
The run with MIO used mio=1000mb

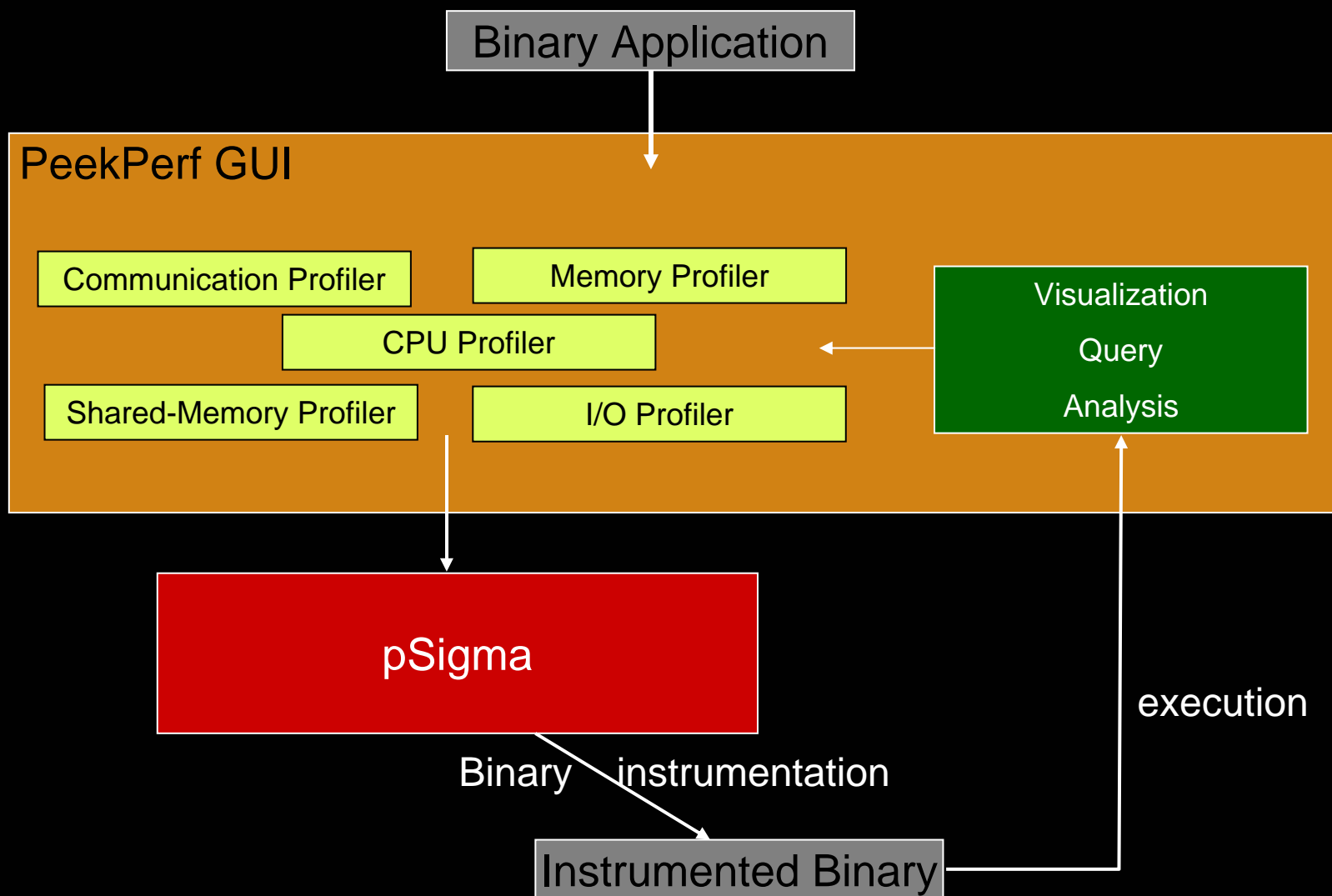
6.8 TB of I/O in 26666 seconds is an average of about 250 MB/sec

The IBM HPC Toolkit

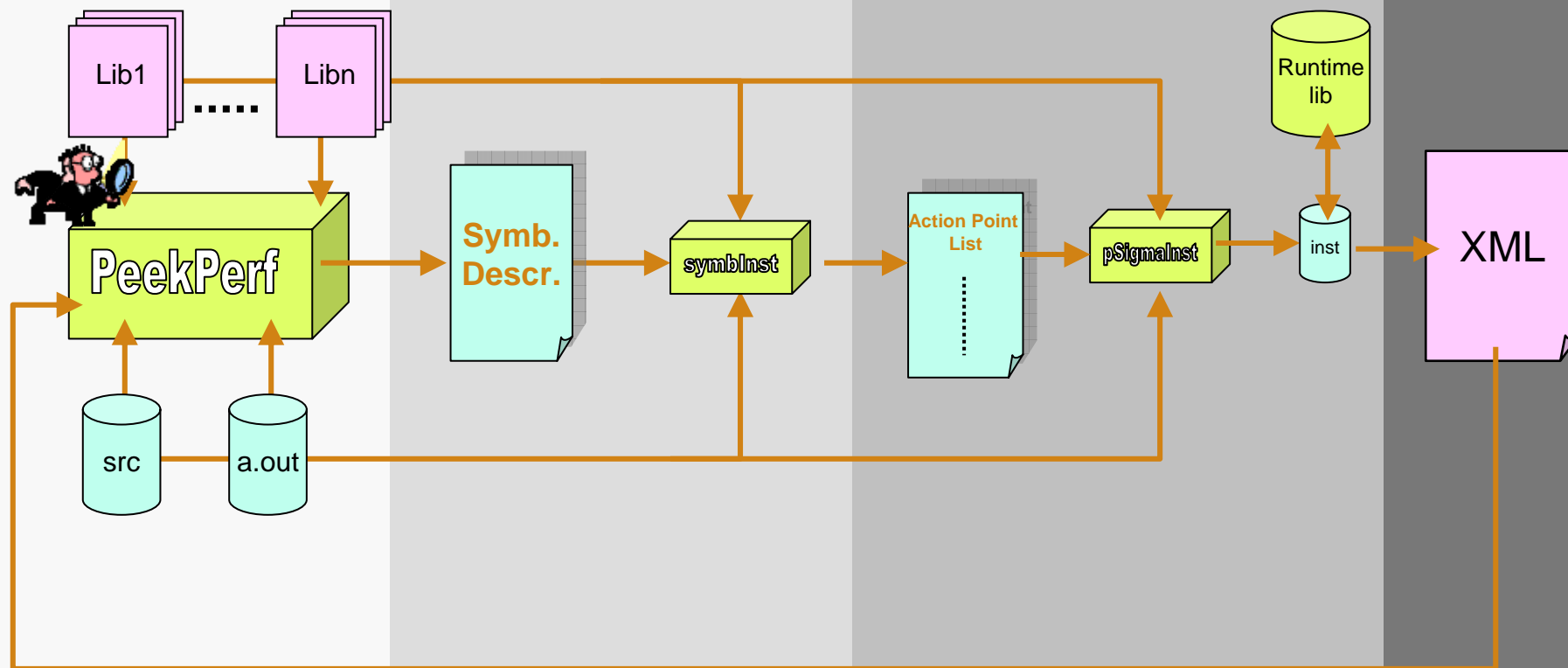
IBM HPC Toolkit

- Instrumentation at the binary level
- Centralized GUI for instrumentation and analysis
- Dynamic instrumentation capabilities
- Enhanced with graphics capabilities (bar charts, plots etc)
- Simultaneous instrumentation for all tools (one run!)
- Query capabilities: compute derived metrics and plot them
- Selective instrumentation of MPI functions

Structure of the IBM HPC toolkit



PeekPerf: Graphical Instrumentation, Visualization and Analysis

Instrumentation**Visualization****Analysis****Symbolic Binary****Instrumentation****Action Point Binary****Instrumentation****Visualization**

IBM ACTC PeekPerf: Main Window

File Tools Options Action

HPM | MPI | SIGMA | DPOMP

HPM_EVENT_SET	60
HPM_DIV_WEIGHT	5
HPM_NUM_INST PTS	100
HPM_WITH_MEASUREMENT_ERROR	1
HPM_MEM_LATENCY	400
HPM_L3_LATENCY	102
HPM_L35_LATENCY	150
HPM_L2_LATENCY	12
HPM_L25_LATENCY	72
HPM_L275_LATENCY	108
HPM_TLB_LATENCY	700

swim_omp

Label	Count	ExcSec	IncSec
-ALL	1	0.006	1.062
-Calc1	102	0.006	0.263
-Calc2	102	0.072	0.373
-Calc3	100	0.098	0.379
-Initial	1	0.042	0.042
-Loop 100	102	0.119	0.119
-Loop 200	102	0.178	0.178
-Loop 300	100	0.209	0.209
-MPI Calc1 end	102	0.08	0.08
-MPI Calc1 start	102	0.05	0.05
-MPI Calc2 end	102	0.068	0.068
-MPI Calc2 start	102	0.055	0.055
-MPI in Calc3	100	0.072	0.072
-loop 110	102	0.009	0.009

```

calc1.f | calc2.f | calc3.f | swim_omp.f | swim_omp.f | calc1.f | calc2.f | calc3.f
70      if(taskid.eq.MASTER)then
71          timei =rtc()
72          WRITE(6,*) 'SPEC benchmark 102.swim'
73
74          WRITE(6,*) ' '
75      endif
76      C
77      C Initializing array req with MPI_REQUEST_NULL
78      C
79      do i=1,16
80          req(i) = MPI_REQUEST_NULL
81      end do
82
83      C      REQUEST PROCESSORS FOR MICROTASKING
84      C SPEC removed CMIC$ GETCPUS 4
85      C
86      C
87      C      INITIALIZE CONSTANTS AND ARRAYS
88      C
89      hpmInit (1)
90      hpmStart (5)
91      CALL INITIAL
92      hpmStop (5)
93
94      if(taskid.eq.MASTER)then
95          WRITE(6,390) N,M,DX,DY,DT,ALPHA,ITMAX
96          390 FORMAT('  NUMBER OF POINTS IN THE X DIRECTION',
97                  1 '  NUMBER OF POINTS IN THE Y DIRECTION',
98                  2 '  GRID SPACING IN THE X DIRECTION',
99                  3 '  GRID SPACING IN THE Y DIRECTION',
100                 4 '  TIME STEP',
101                 5 '  TIME FILTER PARAMETER',
102                 6 '  NUMBER OF ITERATIONS',
103                 //
104                 MNMIN = MIN0(M,N)
105
106      C initial data writes removed for SPEC
107      C      WRITE(6,391) (POLD(I,I),I=1,MNMIN)
108      C      391 FORMAT('/' INITIAL DIAGONAL ELEMENTS OF P ', //
109      C      WRITE(6,392) (UOLD(I,I),I=1,MNMIN)
110      C      392 FORMAT('/' INITIAL DIAGONAL ELEMENTS OF U ', //
111      C      WRITE(6,393) (VOLD(I,I),I=1,MNMIN)
112      C      393 FORMAT('/' INITIAL DIAGONAL ELEMENTS OF V ', //
113      C      DETERMINE OVERHEAD OF TIMING CALLS
114
115      C 6/22/95 for SPEC: JWR: Initialization of TIME
116      TIME = 0
117      NCYCLE = 0
118      t1 = rtc()

```

IBM ACTC PeekPerf: Main Window

File Tools Options Action

HPM MPI SIGMA DPOMP

Name

- [-] Probe
 - [-] MPI_Iprobe
 - [-] MPI_Probe
- [-] Recv
 - [-] MPI_Irecv
 - [-] MPI_RECV
- [-] Send
 - [-] Blocking Send
 - [-] MPI_Bsend
 - [-] MPI_Rsend
 - [-] MPI_Send
 - [-] MPI_Ssend
 - [-] Nonblocking Send
 - [-] MPI_Ibsend
 - [-] MPI_Irsend
 - [-] MPI_Isend
 - [-] MPI_Issend
- [-] SendRecv
- [-] Test
 - [-] MPI_Test
 - [-] MPI_Testall
 - [-] MPI_Testany

swim_omp

Label	Count	ExcSec	IncSec
-ALL	1	0.006	1.062
-Calc1	102	0.006	0.263
-Calc2	102	0.072	0.373
-Calc3	100	0.098	0.379
-Initial	1	0.042	0.042
-Loop 100	102	0.119	0.119
-Loop 200	102	0.178	0.178
-Loop 300	100	0.209	0.209
-MPI Calc1 end	102	0.08	0.08
-MPI Calc1 start	102	0.05	0.05
-MPI Calc2 end	102	0.068	0.068
-MPI Calc2 start	102	0.055	0.055
-MPI in Calc3	100	0.072	0.072
-loop 110	102	0.009	0.009

```

101 1 0,2,MPI_COMM_WORLD,req(2),ierr)
102 CALL mpi_irecv(p(m+1,n+1),1,MPI_DOUBLE_PRECISIO
103 1 0,3,MPI_COMM_WORLD,req(3),ierr)
104 CALL mpi_irecv(vold(m+1,n+1),1,MPI_DOUBLE_PRECI
105 1 0,4,MPI_COMM_WORLD,req(4),ierr)
106 CALL mpi_irecv(uold(m+1,n+1),1,MPI_DOUBLE_PRECI
107 1 0,5,MPI_COMM_WORLD,req(5),ierr)
108 CALL mpi_irecv(pold(m+1,n+1),1,MPI_DOUBLE_PRECI
109 1 0,6,MPI_COMM_WORLD,req(6),ierr)
110 endif
111 if(taskid.eq.0)then
112 CALL mpi_isend(v(1,1),1,MPI_DOUBLE_PRECISION,
113 1 numtasks-1,1,MPI_COMM_WORLD,req(7),ierr)
114 CALL mpi_isend(u(1,1),1,MPI_DOUBLE_PRECISION,
115 1 numtasks-1,2,MPI_COMM_WORLD,req(8),ierr)
116 CALL mpi_isend(p(1,1),1,MPI_DOUBLE_PRECISION,
117 1 numtasks-1,3,MPI_COMM_WORLD,req(9),ierr)
118 CALL mpi_isend(vold(1,1),1,MPI_DOUBLE_PRECISION
119 1 numtasks-1,4,MPI_COMM_WORLD,req(10),ierr)
120 CALL mpi_isend(uold(1,1),1,MPI_DOUBLE_PRECISION
121 1 numtasks-1,5,MPI_COMM_WORLD,req(11),ierr)
122 CALL mpi_isend(pold(1,1),1,MPI_DOUBLE_PRECISION
123 1 numtasks-1,6,MPI_COMM_WORLD,req(12),ierr)
124 endif
125 if(taskid.eq.0.or.taskid.eq.numtasks-1)then
126 CALL MPI_WAITALL(12,req,istat,ierr)
127 endif
128 call f_hpmstop( 17 )
129
130 C
131 RETURN
132 END
133
134 SUBROUTINE CALC3Z
135 C
136 C TIME SMOOTHER FOR FIRST ITERATION
137 C
138 PARAMETER (N1=513, N2=513)
139
140 include "mpif.h"
141 COMMON/decomp/js,je,taskid,numtasks,req(16),
142 1 istat(MPI_STATUS_SI
143 integer taskid,req
144 COMMON U(N1,N2), V(N1,N2), P(N1,N2),
145 * UNEW(N1,N2), VNEW(N1,N2),
146 1 PNEW(N1,N2), UOLD(N1,N2),
147 * VOLD(N1,N2), POLD(N1,N2),

```


Summary

- **The IBM HPC Toolkit is the IBM environment for HPC application performance analysis**
 - Years of field testing and experience
 - Visual source code traceback for all metrics
 - Common framework for simultaneous performance analysis of CPU, MPI, OpenMP, Memory and I/O
 - Available across IBM HPC servers (pSeries, Linux, Blue Gene)
- **Future infrastructure characteristics**
 - Instrumentation of the binary
 - Common instrumentation and analysis GUI
 - Dynamic instrumentation capabilities



IBM Research

Questions / Comments